Review of Source Code Plagiarism Detection Techniques

***Abstract*— In the educational sector, where scientific publications and articles are concerned, plagiarism detection systems are critical. Plagiarism occurs when someone copies a piece of content without the author's permission or citation. You'll need a lot of knowledge about plagiarism types and classes to detect such conduct. Thanks to recently developed tools and procedures, many types of plagiarism may now be recognized. Plagiarism detection has become a crucial concern as a result of advancements in information and communication technology tools (ICT Tools), as well as the availability of online scientific materials. Plagiarism detection has become a crucial issue with the availability of several software text editors. Plagiarism detection and the various types of plagiarism detection datasets used in identification systems has already been the subject of numerous research investigations. This paper discusses various types of plagiarism and various source code plagiarism detection tools developed. Despite the fact that there has been extensive study into systems for detecting source code plagiarism for many years, there is still a need to investigate a robust method.**

***Index Terms*— Plagiarism, Source Code, Programming Course, Software**

I. Introduction

One of the most common errors students make in computer science classrooms is plagiarizing programming assignments. (Mišic, 2017). Plagiarism occurs when someone else's writing is duplicated without the original author's permission. Plagiarism can occur in a variety of formats, including words, programming programmes, computer applications, photographs or drawings, electronic material, and other sorts of work. Plagiarism is one of the most serious global problems that advertisers, researchers, and educational institutions face (Cosma, 2008). Plagiarism is not only deceptive, however, because tasks and homework assignments account for a large amount of the final grade, it tarnishes the assessment process's reputation. Any intentional or unintentional submission and reuse of source code is referred to as source code plagiarism that fails to correctly acknowledge the contributions of others in an academic setting (Cosma, 2008). Plagiarism detection in programming projects varies from the identification of software copies in that it distinguishes between several solutions for small-scale software, which often instead of the thousands of lines of code seen in corporate solutions, consist of hundreds of lines of code. Furthermore, because the majority of students enrolled in programming programmes are inexperienced programmers, some structural and/or other similarities apply (Yang, 2014). A teacher must verify the entries for plagiarism. The fact that these submissions are almost always the same in terms of style and intricacy simply adds to the confusion. Undergraduate students, on the other hand, are frequently exposed to the same instructional resources and training (Simon, 2019), allowing them to derive solutions to identical problems from these materials. When the same resources are employed to direct learning, the options available to a learner are constrained. Plagiarism in source code is a heinous conduct that jeopardises academic integrity. The technique of manually searching for plagiarism-related faults in every programming project. Plagiarism in source of the program or code is a dishonest activity that puts academic integrity at risk. Examining each programming assignment for plagiarism issues on a case-by-case basis is extremely difficult and time-consuming for the instructor. As a result, teachers in programming classes may give students derogatory grades. Plagiarism in source code is trivial to commit but difficult to detect. When a student plagiarises someone else's source code and submits it to their programming instructor, they are committing source code plagiarism. The students then have the option of either reusing the complete copy of the programming assignment or making changes to it, such as using code transformations. Plagiarism in software is a significant issue in programming projects (Arwin C., 2006). This type of incident is most likely to occur in the subject of computer science. Many students repeat other students' project work or programming assignments, or they modify the names of variables or the names and values of methods, making manual plagiarism detection in source code difficult and time-consuming.

II. Plagiarism Process

Collection, inspection, validation, and investigation are the four important processes that must be completed in order to build and implement a robust and error-free plagiarism detection procedure (Osman 2012, Culwin 2001):

A. Collecting information

The plagiarism detector gets the users' required material via a search engine that acts as a route between them and the detector in this first step.

B. Post-Extraction

The detector employs an analysis tool to check for connections in the records to expose the original copy after removing the material (scientific papers, tasks, and other softcopies).

C. Confirming the copy:

A Plagiarism conformation function is required at the analyzing step to distinguish the plagiarized content from the original. This approach may also be used to verify the plagiarized text's degree of plagiarization.

D. Investigation:

If plagiarism is detected, this is the final step, and it is heavily reliant on the user's interference. To distinguish between legitimately plagiarized articles and those that have been quoted, the user's skill is also required.

III. Classification of Plagiarism

Plagiarism can be divided into two categories: monolingual and cross-lingual. Most plagiarism detectors are designed to detect monolingual plagiarism, which occurs when two or more languages are used in the same sentence, such as English

DOI: http://doi.org/10.24086/cocos2022/paper.658

language copy–English language copy. When two or more languages are employed, such as English language setting–Chinese language setting, which is rather unusual (Chowdhury 2018, Dejan 2009, Yang 2009), cross-lingual plagiarism happens. In the following part, the many types of plagiarism will be discussed in depth.

A. Plagiarism Types

Plagiarism can be found in a wide range of works, documents, academic papers, and scholarly publications. a) Claiming someone else's work as your own; b) duplicating someone else's work without attribution or citation; c) appropriating someone else's work and passing it off as your own, whether or whether citation is given, d) misrepresenting another's work as your own by reconstructing it, and e) inserting a false acknowledgement of another's work as your own are all examples of how it can be graded (Anderson, 2011). As previously stated, the two most prevalent types of plagiarism are source code plagiarism and textual plagiarism (N Charya 2015, C. Barnbaum 2009).

B. Textual Plagiarism

Plagiarism in science and scientific-related fields is very widespread, with the full book or article being copied without citing the author or providing a quote. It's also broken down into seven sub-categories, as seen below (N Charya 2015, C. Barnbaum 2009):

1) Copy-paste plagiarism:

This is accomplished by replicating the original text as if it were your own work, without mentioning the authors or the original document.

2) Plagiarism rephrased:

This is divided into two types: a) basic paraphrasing, in which the original text is interpreted in a new way by substituting similar terms from the same context, and b) The text is made by integrating diverse contributions from various articles in mosaic/hybrid/patchwork paraphrase and presenting them in a unique way without referring to the works' initial citations.

3) Metaphor plagiarism:

This is the act of more effectively communicating another's thoughts.

4) Plagiarism of ideas:

When someone else takes your entire method and ideas and presents them as a unique research paper, this is called plagiarism.

5) Recycled plagiarism:

When creators republish their past researches and publications, this is known as republishing.

6) 404 Errors/Illegitimate Source Plagiarism:

This is where the work has been improperly credited.

7) Re-tweet plagiarism:

The term "in-text citation" refers to when the reference is mentioned but no structural, syntactic, or lexical distinction is made between the person's work and the original material.

C. Source-Code Plagiarism

This is most commonly used in educational settings, when someone writes the computer code for a programme, which is subsequently partially or totally reproduced, updated, or reused by another. It is divided into four sub-classes, as listed below (N Charya 2015, C. Barnbaum 2009):

1) Manipulation plagiarism:

This happens when other developers extract or add sub-codes to an existing source code without referencing the citation or recognition.

2) Plagiarism involving reordering structure:

When functions or comments are recorded without referencing the original job, the grammar of the source code is altered.

3) Plagiarism with no changes:

Instead of modifying the code, developers add or delete as part of their job, they may be required to fill in blanks or provide remarks.

4) Language flipping plagiarism:

When a source program code is rewritten in another language and then deemed original source code, this happens.

IV. Plagiarism Detection Methodologies: A Quick Overview

A quick review of some of the strategies used for text document or source code plagiarism detection that have been investigated and analyzed for this research is provided in this part. Despite the fact that many existing technologies cannot be precisely defined as attribute-counting- or structure-based, this section makes a deliberate attempt to categorize these techniques as either attribute-counting- or structure-based. Plagiarism detection tools are divided into three categories:

DOI: http://doi.org/10.24086/cocos2022/paper.658

A. Techniques Based on Attribute Counting

1) Software-metric-based comparison

2) Vector Space Model-based 2-dimensional analysis (VSMs)

a) Semantic Latent Analysis (LSA)

b) Analysis of the Principal Components (PCA)

c) ICP Analysis (Independent Component Analysis) (ICA)

3) Discrete Wavelet Transform Analysis (DWT)

4) Analysis on several dimensions

a) Analysis of temporal latent semantics

b) Analysis of parallel factors (PARAFAC)

B. Techniques Based on Structure

1) Fingerprint and string matching

a) Matching text strings

b) Matching token strings

2) Tree-based detection

a) Abstract Syntax Tree (AST) matching

b) Suffix tree matching

c) PDT-based detection

3) Graph matching

a) Matching of the Program Dependence Graph (PDG)

b) Matching of the Call Graph

Attribute-counting was used in previous ways to find similarities in documents and get similar documents corresponding to a query. When it pertains to identifying duplication in software, some of these algorithms use software measurements, while others use vector space models for texts (Salton, G., 1975). LSA (Deerwester, S. 1990), PCA (Jonathon Shlens 2003), and ICA (Comon, P. 1994, Hyvarinen, A., 2001, Kolenda, T., 2001) are examples of techniques used on VSM for documents where each document is represented as a point in an n-dimensional term-space or a n These approaches convert the input documents into a new vector space, allowing for a more accurate identification of document similarities and differences. Any data manipulation like this tends to bring up hidden patterns in the data.

Both text files and source code files have been used to identify plagiarism using these techniques. Two-dimensional analysis methods like as LSA, PCA, and ICA split a two-dimensional matrices into two or three matrices as a product with distinct characteristics. Singular Value Decomposition (SVD) is used to breakdown the original data matrices into three submatrices (Forsythe, G.E., 1977). Singular vectors and singular values can be found in the component matrices. Instead of working on the original data matrix, PCA preprocesses it and then utilizes Eigen Value Decomposition (EVD), with the component matrices including the covariance matrix's eigenvectors and eigenvalues. SVD can also be used to perform PCA. To split the input data matrix into constituent matrices, ICA uses more complicated measurements. LSA is the most widely used and studied of the three approaches for usage in search engines and plagiarism detection programmes for documents and source code. PCA and ICA have also been used to retrieve documents and detect source code plagiarism. LSA was employed in the earliest stages of this study.

The query is processed by LSA using the resulting factor representation. The dot product of two documents determines their similarity. Another method, temporal LSA, which incorporates a third dimension of time into LSA, has been used to test the impact of temporal information in document retrieval. Parallel factor analysis (Harshman, R.A. 1970, 1984a), a multi-dimensional factor analysis technique, has also been employed for document indexing (Harshman, R.A. 1984a). Another method for document retrieval that has been used is DWT (Park, L.A.F., 2005b). Unlike vector space models, DWT-based document retrieval helps to keep positional information and also allows for analysis at multiple levels of resolution depending on the wavelet and decomposition level chosen. Structure-based strategies are favored over attribute-counting-based methods for detecting source code plagiarism. Text string matching, token string matching, and graph or tree matching are the most common. Exact and inexact text string matching methods are available. Each source code file can be treated as a simple text file for accurate matching. The files are then compared line by line, with each line treated as a string of characters. To locate exact matches, a character-by-character comparison is performed. Substring matching works by splitting the text into substrings and then comparing them. Inexact matches can also be found by combining substring matching with hashing. Transform each file into a single string of characters and divide it into fixed-length substrings. Calculate the hash value for each substring. Group the substrings so that all of them have the same hash value and are placed in the same hash bucket. To find code matches, compare substrings whose hash function will be the same. Another method is to hash each line of the source code file and then compare lines that have into the same hash bucket. To find matches between strings, sequence matching algorithms have been utilized for both global and local alignment. For global sequence alignment, the Needleman-Wunsch (NW) algorithm (Needleman, S.B 1970) and the Longest Common Subsequence (LCS) algorithm Hirschberg, D. S. 1977, Cormen, T.H. 1990, Gonnet, G. H. 1991) are used, while the Smith-Waterman (SW) algorithm (Smith, T.F. 1981) is used. Token string matching algorithms have two phases: a transformation phase in which source code is changed into a new internal representation and other is a comparison phase in which transformed codes are compared using various algorithms. Tokens are generated after scanning all of the source code files. Token sequences are formed by traversing the tokenized code, and these token sequences are matched using sequence matching algorithms. All file-pairs with scores larger than a similarity criterion are flagged as plagiarized. AST matching, suffix tree matching, software dependency graph matching, and other graph or tree matching algorithms are examples. Suitable parsers are used to create ASTs of the source code files to be compared. AST matching can be done in a variety of ways. The tree can be traversed to generate node

DOI: http://doi.org/10.24086/cocos2022/paper.658

sequences, which can then be compared using sequence matching methods. Subtree hashing is another option. Hash each subtree after dividing the AST into subtrees. To locate similar subtrees, group subtrees in the same bucket with the same hash value and compare them. In a similar way, suffix trees, programme description trees, programme dependence graphs, call graphs, and other graph or tree representations of source code can be used to find code matches.

Many authors have tried a variety of ways to find a highly accurate plagiarism detection method, but due to advancement of technology, algorithms, and data mining applications, it has remained difficult to find the right one. As plagiarism detection systems have advanced, however, this trend has become a double-edged sword. In reaction to the prohibition of illicit techniques of copying researchers' initial work, technologies for detecting this type of theft have evolved (Dejan 2009, Yang 2009). In the following sections, many methodologies and strategies for detecting plagiarism will be discussed.

V. Methods for Detecting Plagiarism

Plagiarism has been a serious worry in the scientific world, and researchers have employed a range of approaches to counteract it (Chowdhury 2018). As a result, Table I presents a comparative of plagiarism detection from the standpoint of academics.

It's worth noting that nearly every one of these forms tries to match a current document with a question document (Chowdhury 2018):

1) Methods based on characters:

The n-gram and word n-gram approaches, which are based on the string-matching methodology, are used to determine the degree of string matching/mismatching (C. Grozea 2009, C. Basile 2009).

2) Vector based method:

Tokens, rather than strings, are used to perform lexical and syntax functions in the vector-based technique (H. Zhang 2011).

3) Parts of speech (POS),

Verbs, adverbs, nouns, pronouns, adjectives, prepositions, conjunctions, and interjections are used in this approach of detecting plagiarism in a text document (M. Elhadi 2008).

4) Semantic-based approach:

This method compares the similarity of two words in a letter to find semantic similarities. To maintain the semantics of two different statements, move from active to passive voice (C. Fellbaum 2009, Resnik 1999).

5) Machine Learning Based method:

This method employs machine learning, and the sentences are translated into numerical or character values. It generates a 0 or 1 result to identify plagiarism, with 0 indicating that the papers are uniquely different (no plagiarism) and 1 indicating that they are similar (plagiarism is found) (R. Yerra 2005, J. Koberstein 2006, S. M. Alzahrani 2009).

6) Structure based method:

The focus of a structure-based approach is on how in a document, words are written in a specific block of text (M. Rahman 2007, T. W. Chow 2009).

7) Stylometric based method:

To detect plagiarism, employ a stylometric technique. This kind attempts to determine the author's writing styles by discovering correlations between two blocks of possible to compile on the stylistic features of the author (S. M. Zu Eissen 2007, E. Stamatatos 2009).

8) Methods for detecting cross-lingual plagiarism:

When there are so many different ways to plagiarize, such as cross lingual syntax based methods and/or dictionary-based methods, it might be difficult to identify plagiarism. It demands a deep knowledge of different languages utilized in a variety of publications (M. Potthast 2011, H. Osman 2012).

9) Semantic based approach:

It addresses the shortcomings of a semantic-based approach. Method for detecting plagiarism based on grammar and semantics: It's a powerful tool for natural language processing that's extensively utilised (NLP). It's quite good at catching plagiarism that's been copied and pasted or paraphrased (J. P. Bao 2003).

10) Cluster based method:

When compared to other methods, classification and cluster-based strategies are extremely useful in retrieving information while searching for some plagiarized records. Furthermore, when compared to other methods, the matching period during the discovery phase is reduced (V. Mitra 2007).

11) Citation-based approach:

For the use of semantics in the cited text, this is a new method that mostly belongs to semantic plagiarism detection approaches. It searches for similar pairs of papers based on the citation, and these algorithms make advantage of the semantics of the citation (B. Gipp 2010, 2011, Tapan P. Gondaliya 2014).

VI. Source-Code Plagiarism Detection Tools

There are many different types of plagiarism detection methods, each of which can be classified based on their methodologies. Based on Mozgovoy's (M. Mozgovoy 2006) work, this section offers methods for detecting source-code plagiarism. Two examples are fingerprint-based systems and content comparison algorithms. There have been several categories offered in the literature (T. Lancaster 2005, K. Verco 1996a, 1996b).

A. Fingerprint based systems

The fingerprint approach generates fingerprints for each file basis of statistical data such as the average number of phrases per line, the number of unique terms, and the amount of buzzwords. Comparable files have fingerprints that are almost identical. In order to evaluate their proximity, the distance between them is usually calculated (In a mathematical sense, using a distance function) (M. Mozgovoy, 2006).

Attribute counting systems formerly employed fingerprints. Ottenstein created the world's first plagiarism detecting system, which used attribute counting software to detect identical and almost identical student work (K. Ottenstein 1976a, 197b). The tool employed Halstead's software measures for detecting duplication in ANSI-FORTRAN packages by counting operations and operands (M. Halstead, 1972, 1977). Halstead

DOI: http://doi.org/10.24086/cocos2022/paper.658

proposed the following metrics:

- total number of operator occurrences
- the number of distinct operands
- the amount of distinct operators
- total number of operand occurrences

Soffa and Robinson in (S. Robinson, 1980) produced plagiarism detection software that combined new measures with Halstead's measures to improve detection. ITPAD was a three-step process that included lexical analysis, programme structure analysis for characteristics, and characteristics analysis. The tool termed as Institutional Tool for Program Advising (ITPAD) breaks down each programme into sections and graphs the programme structure. The computer then analyzes programmes by generating a list of features based on the lexico - grammatical analyses by counting these properties.

Rambally and Sage (G. Rambally, 1990) created an attribute counting system that parses student programmes and generates a knowledge system with knowledge vectors, each of which contains information on the attributes in the student's programme. They use the same properties found by the preceding techniques, but they count the loop-building attributes within a programme in a different way. Rather than counting versions of loop-building statements separately or aggregate the amount of repetitions of all of these sentences into the a target variable count to differentiate between various types of loops.The programmes are categorized in a decision tree after the knowledge vectors have been constructed. The decision tree is used to identify the programmes that have commonalities.

More advanced plagiarism detecting tools have been created since then. Structure metric systems are a term used in the literature to describe these instruments. Structure metric systems compare programme structures to see how comparable they are. These are classed as content comparison techniques by Mozgovoy (M. Mozgovoy, 2006). A comparison of attribute-counting and string-matching algorithms indicated that attribute-counting methods are insufficient for identifying plagiarism on their own (K. Verco 1996a, 1996b, G. Whale, 1990). A more recent plagiarism detection method, MOSS (Koschke, R. 2007), combines fingerprinting and structural metric approaches.

MOSS (Koschke, R. 2007, S. Schleimer 2003) is a string matching algorithm that k-grams are being used to break down programmes each of which is a k-length unbroken segment. MOSS uses a set of the hashes to serve as the program's fingerprints after hashing each k-gram. The number of signatures shared by two programmes indicates their similarity; the more fingerprinting they share, the more comparable they are. The number of tokens matching, the number of points matched, and the proportion of source-code overlapping between the detected file pairs are even included in the results report for every pairing of identified source-code chunks (Schleimer, S. 2003).

B. Content comparison techniques

Structure-metric systems are sometimes used to refer to content comparison techniques in the works. Such systems turn algorithms into tokens and afterwards look for contiguous substring patterns inside the programmes that satisfy. How similar two programmes are measured by the percentage of text that matches. Content comparison methodologies were classified by Mozgovoy (M. Mozgovoy, 2006) as:

- parameterized matching algorithms,
- String matching algorithms, and
- parse tree comparison algorithms

1) String matching algorithms

The structure of programmes is compared in most contemporary plagiarism detection algorithms. To improve plagiarism detection, these methods compare the programme structure as well as attribute-counting measures. The comparison techniques used by string-matching systems are more difficult than those used by attribute-counting systems. The majority of string corresponding algorithms work by converting programmes into tokens and then using a smart search approach to locate conventional text subsequence between two projects. Donaldson et al. (J. Donaldson 1981), who found basic ways that beginner programming students employ to mask plagiarism, were the first to propose such systems. These tactics include renaming variables, reordering statements that have little bearing on the program's conclusion, breaking up expressions, such as numerous declaration and output statements, and altering formatting assertions.

Donaldson et al. (J. Donaldson 1981) have highlighted some of the most fundamental modes of attack. A more complete list of attacks can also be found in Whale (G. Whale, 1999) and Joy and Luck (M. Joy, 1999). Prechelt et al. (L. Prechelt 2002) present a considerably more comprehensive list. We also present a comprehensive list of plagiarism assaults, which may be found in (G. Cosma 2006).

Donaldson et al. (J. Donaldson 1981) created a tool that analyses software components for particular kinds of statements and keeps a record of those. Then a single code character is assigned for statement types that are relevant in characterising the structure. Each assignment after that is represented by strings. If representations of such character strings are similar, the pair of programmes is considered as plagiarized parts of the software elements.

Plague (M. Mozgovoy 2007), YAP3 (Yet Another Plague) (M. Wise (1996), JPlag (L. Prechelt 2002), and Sherlock (Mozgovoy, M. 2005) are some current string-matching-based systems.

Tokenization is the first step for most pattern matching systems, including the ones mentioned above. Each data sample is substituted with pre-defined and unambiguous tokens during the tokenization process; For example, regardless of loop type, numerous loop kinds in the code may be substituted by same tokens name. (e.g. for, while etc.). As a result, each tokenized code is represented as a series of tokens. After that, the programmes are evaluated by looking for similar tokens fragment patterns. Comparable programmes have a comparable number of tokens above a given threshold. The most common method of calculating similarity between two files is the

DOI: http://doi.org/10.24086/cocos2022/paper.658

percentage of tokens in the detected files that are matched.

Plague (M. Mozgovoy 2007) generates a set of tokens for each program and then uses a pattern matching approach to evaluate the tokenized versions of selected programmes. The values are presented as a list of matching pairings, with the duration of the matching element of the tokens strings between both the files being ranked first. The plague is known for causing a wide range of issues (P. Clough 2000). One of several issues with Plague is that converting to another programming language takes a very long time. Still, the results are presented in two lists, each of which is ordered by indices, making understanding of the data difficult. Finally, Plague is inefficient since it requires several Unix tools, causing portability concerns.

The token matching mechanism is used by YAP3 to turn programmes into a string of tokens and compare them. To locate similar source code portions, use the Running Karp Rabin Greedy String Tiling algorithm (RKR-GST) (M. Wise 1996). Before converting source-code files to tokens, YAP3 pre-processes them. Eliminating comments, converting capital letters to lowercase, translating analogues to a common form (e.g., functional to procedures), rearranging procedures into their call order, and removing all tokens not present in the destination syntax lexicon. (i.e., eliminating any terms that aren't reserved in the language). YAP3 was designed to find the reordering of independent source-code segments and the splitting of code functions into several functions. Several strings are compared by the method (the pattern and the content), scanning the text for matching pattern substrings. Tiles are used to describe substring matching. Each tile is a match that includes both a pattern and a text substring. The status of tokens within the tile is set to mark when a match is detected. Tiles with a matching size less than the minimal are ignored. The RKR-GST approach attempts to optimize the token represented by tiles by finding maximum matching of continuous segment sequence that comprise tokens which have not been spanned by the other subsequences.

JPlag (L. Prechelt 2002) uses the same comparison mechanism as YAP3, however it is faster to run. In JPlag, similarity is calculated based on the fraction of token strings covered. One of JPlag's flaws is that files must parse in order to be included in the plagiarism comparison, resulting in the loss of comparable files. The user-defined minimum-match-length option in JPlag is also set to zero. Modifying this value can have a positive or negative impact on the detection results, and changing it may necessitate knowledge of the JPlag algorithm (i.e RKR-GST). JPlag is a web service that has a simple yet effective user interface. The user interface includes a comparison display that emphasises matched source-code fragments in the detected comparable files, as well as a list of similar file pairs and their degree of similarity.

In addition to YAP3, Sherlock (Mozgovoy, M. 2005) uses a similar approach. Sherlock turns programmes into tokens and looks for common line sequences (called runs) in two files. Sherlock, like the YAP3 algorithm, looks for the longest possible runs. Sherlock's new interface shows a list identifying file pairings that are related, their reference standard, and corresponding blocks of source-code snippets identified within those document pairings. Sherlock also creates a network, for each vertices indicating a specific source file so each edge indicating how closely interrelated files are. Only files that are comparable (i.e., have edges) beyond a user-defined threshold are shown in the graph. One of it's features of Sherlock seems to be that, like JPlag, it's doesn't need the documents to be processed before even being analyzed, but it does not have any user-defined parameters that can slow down the whole system. Sherlock is a free, open-source application that uses a token similarity score that may be simply adapted to different languages (Mozgovoy, M. 2005). Sherlock is a standalone application, unlike JPlag and MOSS, which are both web-based services. When it comes to screening student files for plagiarism, a stand-alone application may be preferred for academics due to confidentiality concerns.

Plaggie (A. Ahtiainen 2006) is a tool similar to JPlag, however it uses the RKR-GST instead of performance optimization techniques. Plaggie turns files into tokens and compares them for resemblance using the RKR-GST method. Plaggie's goal is to create a stand-alone (that is, placed on a single desktop) variant of JPlag that enables academics to omit software from the evaluation, such as software supplied to students in the classroom.

Mozgovoy et al. developed another plagiarism detection method, the Fast Plagiarism Detection System (FDPS) (M. Mozgovoy 2005, 2007a). By storing files in an indexed data structure, FDPS attempts to increase the speed of plagiarism detection. Tokenized files are kept in an indexed data structure after being turned into tokens. Using an approach similar to that of YAP3, this enables for quick file searching. This entails extracting substrings from a test file and checking the collection file for a match. The matches are saved in a repository and then utilized to determine how similar two files are. The ratio of the total tokens matching in the collecting record to total tokens in the test dataset is known as similarities. The distribution of matched tokens is determined by this ratio. The document pairings that have a resemblance value greater than just the criterion are found. The outcomes of the FDPS tool cannot be viewed simply showing comparable code chunks, which is one of its drawbacks (M. Mozgovoy 2007b). As a result, the authors used FDPS in combination with the Plaggie programme to compare files and show the results.

2) Parameterized matching algorithms

Standard token methods for pairing are identical to parameterized matching algorithms, except they require a more complicated tokenizer (B. Baker 1995a). To begin, these parametric matched methods automatically convert to tokens. A parameterization matching (sometimes called as just a p-match) analyzes source-code fragments with deliberately altered prefixes (renamed).

The Dup tool (B. Baker 1995b) was created to detect duplicate code in software using a p-match technique. It recognises parts of source code that are identical or parameterized. The tool first analyses the source-code file using a lexical analyzer, creating a converted line for each line of code. The original code is converted into arguments by converting variables and constant

DOI: http://doi.org/10.24086/cocos2022/paper.658

into same symbols P, and a list of parameters possibilities. The line x = fun(y) +3x, for example, is transcribed into P = P (P) +P p, resulting in a list of x, fun, y, 3, and x. The parser produces a number for each altered piece of code. Dup seeks to identify source-code p-matches between files given a threshold length. The fraction of p-matches between two files is used to determine similarity. The Dup software computes a proportion of between both files, a profiles of p-matched programming language, and a graph shows the location of the matching software. If fraudulent statements are supplied or a small expression evaluates to true be rearranged, the Dup software refuses to discover documents, as per the application (D. Gitchell 1999). There are various parameterized matching algorithms in the literature (A. Amir 1994, B. Baker 1995a, R. Idury 1994, L. Salmela 2006, K. Fredriksson 2006).

3) Parse Trees Comparison Algorithms

Parser graph comparing methods analyze the architecture of documents using comparative methods (B. Baker 1995a). Each file is represented as a parse tree using the SIM tool (D. Gitchell 1999), which then compares file pairs using string representations of parse trees. Every code will be first transformed into a sequence of tokens, for each token substituted by a set of predefined of tokens. The token sources of the documents are separated in parts and these segments are matched following tokenization. This approach can be used to find code fragments that have been mixed together.

SIM compares file pairs expressed as parsing tree using basic clustering methods. SIM, like most pattern matching techniques, looks for most frequent token substring. SIM was created to identify similarities in C programmes and to calculate the measure of correlation among set of files (in the range of 0.0 to 1.0). SIM's developers hope it will be simple to modify it to operate with writing programs in language apart from C. It can detect files that have undergone comparable changes, including such new names, reordering statement and function, and going to add remarks and white spaces, among many other features (A. Flint 2006).

The Brass system (B. Belkhouche, 2004) implements parse tree comparison algorithms in a more modern way. Every programme can be displayed as a flowchart, which would be a visual illustration of each file as a tree, thanks to Brass's ability. The document header is specified at the root of the tree, while the instructions and controlling mechanisms present in the file are specified at the subtree. After that, the tree is converted into binary trees, and a symbolic tables (data dictionaries) is created, which includes data on the parameters and datatypes used in the documents. A three-step procedure is used to compare brass. The first two methods compare files' structural trees, whereas the third algorithm compares the data dictionary representations of the files.

Tree comparison necessitates the use of more complicated algorithms than string matching techniques. As a result, systems using tree comparison techniques are expected to be slower than systems using string matching methods (B. Baker 1995a). Brass' filtering technique accelerates the comparison process.

There hasn't been much research into parse tree algorithms, according to Mozgovoy (M. Mozgovoy, 2006), and it's unclear whether in order to detect linked source-code file pairings, they outperform clustering techniques.

VII. Comparison of Tools and Methods

Scientists have created a number of approaches for detecting open source duplication, including:

- A system software recognition application that is used in source plagiarism detection to provide a clearly associated with academic and remove the shown outcome. It was created to interact with the computer programming LISP and Pascal. For a stretch of 100 LISP, this tool took a respectable length of time (S. Dumais 1988).
- A system that is primarily used to detect four types of code source plagiarism: changing the code name, rebuilding or recoding the code, moving, adding, changing, and removing the code, and replacing some text with the code. A bottom-up technique was used to accomplish six phases:

  1- First, they pre-flatter the source code by marking and renaming each alphanumeric string in the code, which is a popular way for filtering source code.

  2- Secondly, they divide the code into sections and evaluate how identical the fragments are.

  3- Researchers next compared each component before re-posting it for screening.

  4-5: Finally, in the examination of the document, the usage of matrix M that has been utilized in filtering.

  6- At this point, begin analyzing the original document based on the document-by-document distance evaluation. This strategy has been used on copra languages with excellent success (Salton, G., 1975).
- A system for Java that detects plagiarism in source code (PDE4Java). Three phases make up the suggested search engine.

  1- Tokenization is the first stage in the Java code tokenization process.

  2- Inside the phase two, locate and evaluate the similarities between both the source code and the tokenized script.

  3- Finally, the Java code must be clustered in order to be used as a reference in plagiarism detection. Because of its versatility, this search engine can be utilized with any computer language. Aside than the textual (L. Prechelt 2002), a report can be generated for each cluster code. Figure 1 shows some of the existing popular tools to detect plagiarism in source code.

DOI: http://doi.org/10.24086/cocos2022/paper.658

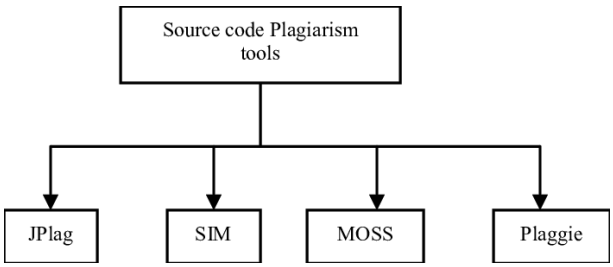

Fig. 1. Some of the popular tools

More such systems are discussed in sections given below:

A. MOSS

MOSS (Koschke, R. 2007, S. Schleimer 2003) is a software plagiarism detection programme that uses source code similarity to detect software plagiarism. It's offered as an online service: documents may be uploaded using a script, and the results can be viewed after processing using a web interface. MOSS compares documents using character level n-grams (a continuous subsequence of length n). For the sake of efficiency, only some of the features are compared instead of all of them. Calculating a hash value for each feature and selecting only a subset of those features using 0 mod p for a fixed p is a typical technique for selecting textual features. The authors point out that this technique frequently leaves gaps in the papers, increasing the chances of missing matches between them. They utilize a winnowing approach to prevent this from happening: instead of randomly selecting n-grams from the page, they select at least one characteristic for each window. They also employ a big n value to avoid noisy findings and delete white space characters to avoid white space matching.

B. JPlag

JPlag (L. Prechelt 2002) is a tool for sorting programmes based on their similarity. The authors claim that comparing programmes just on the basis of a feature vector ignores structural similarity far too much. Rather, they attempt to match on what they refer to as structural traits. It start by converting the Java source into a series of tokens, such as BEGINCLASS, ENDCLASS, and BEGINMETHOD and ENDMETHOD, before using the text. Then, using the list of tokens, they employ an algorithm to locate matches between documents, working from the largest to the smallest matches. Small matches would occur far too frequently if there was no setting for the minimum size of matches. They use the Karp-Rabin (M. Mozgovoy, 2006) technique to add a few runtime optimizations to the basic comparison procedure, which has a worst case O (n3) times complexity (where, n gives sizes of document). They compare alternative cut-off criteria and methodologies for determining a threshold value based on dataset similarity values. It also run some measurements on datasets that examines the impact of the minimal matching length and the set of tokens to utilize. They also show potential possible JPlag assaults.

C. Sherlock

Sherlock (Mozgovoy, M. 2005) is a simple C programme that uses similarities to arrange text document pairs like source code. The software starts by creating signatures. It drops whitespace characters and a percentage of the other characters from the text file in a fairly random manner when producing the signature. After that, all of the signatures are compared to one another.

D. Plaggie

Plaggie (A. Ahtiainen 2006) is another tool that allows you to compare Java source code documents for similarities. It functions similarly to JPlag. The key distinctions at the time of publishing were that Plaggie was open source and could be operated locally. JPlag is currently available as open source software that can also be used locally.

E. SIM

SIM (L. Prechelt 2002, S. Dumais 1988) is a C-based programme and plagiarism detection tool for text. It works by first tokenizing the files and then scanning the file pairings for the longest common subsequence.

F. Marble

Marble (Salton, G., 1975) is a tool that was created with the goal of being as simple as possible. Normalization, sorting, and detection are the three phases of the tool. The normalization process transforms source code into a more abstract programme from raw text. Class, extends, and String keywords are preserved, but names are changed to X and numeric literals to N. Symbols and operators are also left in place. In this transformation, import declarations are removed. Classes and class members are lexicographically sorted after normalization, making the tool indifferent to reordering these programme structures. Finally, using the number of altered lines normalized by the overall length of the two files, the streamlined programme is compared using the Unix line-based diffing utility diff.

G. GPlag

To compare two programmes, GPlag (Deerwester, S. 1990) uses a programme dependence graph (PDG) analysis. The premise is that even after modifying code, dependencies between software sections often remain the same. Following the conversion of the programmes to a PDG, any subgraphs bigger than a "trivial" size are tested on graph isomorphism relaxed with the relaxation parameter $\gamma \in (0, 1)$. If the subgraph $S \subseteq G$ is subgraph isomorphic to G' and $|S| \geq \gamma|G'|$, then graph G is said to be γ-isomorphic to G'. They exclude pairs of graphs that do not have identical histograms of their vertices to avoid examining every pair of sub-graphs.

H. Evolving Similarity Functions

For learning similarity functions, Wu and colleagues (J. Zhu 2015) recommend adopting a genetic approach. The first contribution involves using Particle Swarm Optimization to determine acceptable parameters for the OkapiBM25 similarity function. The default parameters for plagiarism detection are shown to be poor, and can be improved using a genetic optimization approach. The performance of show parameters is similar to that of JPlag; the differences are not significant. The second contribution is to learn similarity functions by utilizing

DOI: http://doi.org/10.24086/cocos2022/paper.658

a Genetic Programming approach to optimize similarity functions. Based on a pool of the best current functions, a genetic algorithm generates new programmes. Simple grammar limits the operations that can be performed, such as addition and multiplication, as well as some endpoints like within-document word frequency, within-query term frequency, and document length. Add a penalty for similarity function size. Three separate fitness functions must be optimized. The authors speculate that this is related to functions returning negative similarity scores as well as overfitting to the training set, as the resulting functions perform worse than other functions.

I. Plague Doctor, Feature-based Neural Network

Anti-plagiarism software Plague Doctor, as described in (S. Engels 2007), employs source code properties as input to a Neural Network model. In addition, the model can calculate the relative relevance of each feature. It uses 12 numerical features such as Moss outcome, comment similarity, misspelled word ratio, and so on. They train a classifier using a tiny Neural Network with seven hidden units utilizing the features. The proportional value of each feature can be determined by evaluating weights in neural network connections. More significant features will have higher weights for their connections after learning than less important features. Moss output has the highest normalized weight (0.2390), followed by the ratio of misspelled comments (0.1418) and string literal similarity (0.1418). (0.1147). The F-measure on the hold-out set is greater than when using Moss alone, although the authors believe this is due in part to the system being trained on that dataset.

J. Callgraph Matching

M. L. Kammer (2011) uses Callgraph Matching as a plagiarism detection approach for the Haskell programming language. Each programme is first converted into call graphs, which are then preprocessed. After that, a subgraph isomorphism method and an edit distance algorithm employing A* searches are employed to identify matches for all subtree pairs. They compare the total similarity scores obtained after applying various types of source alterations to the token-based tools Holmes and Moss, which employ character-level n-grams in conjunction with a fingerprinting technique. On a desktop PC, comparing a set of 59 apps takes around a day (from 2011 or before). To speed up the comparison, the application might make advantage of several threads and machines.

K. Holmes

For Haskell applications, Holmes (J. Hage 2013) is a plagiarism detection tool. The tool compares two programmes using a variety of techniques, including a Moss-like fingerprinting methodology, a token stream, and three alternative ways of comparing degree signatures: based on Levenshtein edit distance of two vertices, and Levenshtein distance of two vertices paired with position. The call graph does not take into account local functions. Holmes does a reachability analysis and, to some extent, eliminates code that is not reachable from the entry point. Most IDs and comments are eliminated totally during the transition. A teacher can add template code to prohibit pupils from matching on code that they are allowed to use.

L. DECKARD Code Clone Detection

Deckard (L. Jiang, 2007) detects code clones in a software project using a tree-based technique. They record information about trees in the programme using characteristic vectors, which are the element-wise sum of occurrences of types of nodes in the programme portion, where each node is an instance of the program's AST (Abstract Syntax Tree). A minimum token count (a minimum $\ell 1$-norm) can be provided to prevent matching on short vectors. The vector does not include the occurrences of some nodes, such as [,], (and). They also integrate programme sub-trees using a sliding window method. The Euclidean distance is used to group similar vectors. They use Locality Sensitive Hashing to reliably map similar vectors to equivalent hash values.

CONCLUSIONS

This study included a thorough review of the literature on plagiarism types, strategies, and tools. The seven types of code plagiarism and the four types of source-code plagiarism were highlighted. Following that, plagiarism detection methods and tools were demonstrated, with newly developed tools being more advanced. The majority of the tools operate online, requiring only data over internet and website pages to function, with some being free and others requiring a subscription fee. Finally, the most common types of plagiarism were discussed, as well as the most difficult aspects of using plagiarism detectors. Despite the fact that plagiarism checking software has been here for a long time, a more comprehensive method is still needed. In the future, we want to use advanced machine learning and artificial intelligence techniques to create a workable model.

REFERENCES

Ahmed Hamza Osman, Naomie Salim, and Albaraa Abuobieda (2012) "*Survey Of Text Plagiarism Detection*" Computer Engineering And Applications Vol. 1, No. 1, June 2012 ISSN: 2252-4274 (Print) 37 ISSN: 2252-5459 (Online)

Arwin C, Tahaghoghi SM. (2006) "*Plagiarism Detection across Programming Languages*". Hobart, Tasmania, Australia: Australian Computer Society, Inc; 2006:277-286.

B. Gipp, J. Beel, (2010) "Citation Based Plagiarism Detection: A New Approach To Identify Plagiarized Work Language Independently", In: Proceedings Of The 21st Acm Conference On Hypertext And Hypermedia, Acm, 2010, Pp. 273--274.

B. Gipp, N. Meuschke, (2011) "*Citation Pattern Matching Algorithms For Citation Based Plagiarism Detection: Greedy Citation Tiling, Citation Chunking And Longest Common Citation Sequence*", In: Proceedings Of The 11th Acm Symposium On Document Engineering, ACM, 2011, pp. 249--258.

C. Barnbaum, (2009) "*Plagiarism: A Student's Guide To Recognizing It And Avoiding It"*. [Online]. [Cit. 2010-12-14].

DOI: http://doi.org/10.24086/cocos2022/paper.658

C. Basile, D. Benedetto, E. Caglioti, G. Cristadoro, M. Esposti, (2009) *"A Plagiarism Detection Procedure"*, In Three Steps: Selection, Matches and Squares, In: Proc. Sepln, 2009, Pp. 19-23.

C. Fellbaum, WordNet: An Electronic Lexical Database DOI: https://doi.org/10.7551/mitpress/7287.001.0001 ISBN (electronic): 9780262272551 The MIT Press, 1998.

S. Torres, A. Gelbukh, (2009) "Comparing Similarity Measures For Original WSD Lesk Algorithm", Research In Computing Science 43 (2009) 155-166. 26. P.

C. Grozea, C. Gehl, M. Popescu, (2009) "*Encoplot: Pairwise Sequence Matching In Linear Time Applied To Plagiarism Detection*", In: 3rd Pan Workshop. Uncovering Plagiarism, Authorship and Social Software Misuse, 2009, P. 10.

C. Leacock, G. A. Miller, M. Chodorow, (1998) "*Using Corpus Statistics And Wordnet Relations For Sense Identification*", Computational Linguistics 24 (1) (1998) 147-165.

D. Sraka and B. Kaucic, (2009) "*Source code plagiarism*," Proceedings of the ITI 2009 31st International Conference on Information Technology Interfaces, 2009, pp. 461-466, doi: 10.1109/ITI.2009.5196127.

D. Pawelczak, (2018) "*Benefits and drawbacks of source code plagiarism detection in engineering education*," 2018 IEEE Global Engineering Education Conference (EDUCON), 2018, pp. 1048-1056, doi: 10.1109/EDUCON.2018.8363346.

D. Santos de Campos and D. James Ferreira, (2020) "*Plagiarism detection based on blinded logical test automation results and detection of textual similarity between source codes,"* 2020 IEEE Frontiers in Education Conference (FIE), 2020, pp. 1-9, doi: 10.1109/FIE44824.2020.9274098.

E. Stamatatos, (2009) "*A Survey Of Modern Authorship Attribution Methods"*, Journal Of The American Society For Information Science And Technology 60 (3) (2009) 538-556

F. Culwin and T. Lancaster, (2001) "*Plagiarism Issues For Higher Education",* Vine, Vol. 31, Pp. 36-41, 2001.

F.-P. Yang, H. C. Jiau, and K.-F. Ssu, (2014) "*Beyond plagiarism: An active learning method to analyze causes behind code-similarity*," Computers & Education, vol. 70, pp. 161-172, Jan 2014.

G. Cosma and M. Joy, (2008), "*Towards a definition of source-code plagiarism,"* IEEE Transactions on Education, vol. 51, no. 2, pp. 195-200, May 2008.

H. Osman, N. Salim, A. Abuobieda, (2012) "*Survey of Text Plagiarism Detection*", Computer Engineering and Applications Journal (Comengapp) 1 (1) (2012) 37-45.

H. Zhang, T. W. Chow, (2011) "*A Coarse-To- Ne Framework To Efficiently Thwart Plagiarism*", Pattern Recognition 44 (2) (2011) 471-487.

Cheers, H., Lin, Y. & Smith, S.P. (2021) "*Evaluating the robustness of source code plagiarism detection tools to pervasive plagiarism-hiding modifications*," Empir Software Eng 26, 83 (2021). https://doi.org/10.1007/s10664-021-09990-4

Hayden Cheers, Yuqing Lin, and Shamus P. Smith (2020) "*Detecting Pervasive Source Code Plagiarism through Dynamic Program Behaviours,"* In Proceedings of the Twenty-Second Australasian Computing Education Conference (ACE'20) Association for Computing Machinery, New York, NY, USA, 21–30 DOI:https://doi.org/10.1145/3373165.3373168

Hussain A Chowdhury and Dhruba K Bhattacharyya, (2018) "*Plagiarism: Taxonomy, Tools and Detection Techniques*", [V1] Fri, 19 Jan 2018 07:27:42 Utc (886 Kb).

J. Koberstein, Y.-K. Ng, (2006) "*Using Word Clusters To Detect Similar Web Documents*", In: International Conference On Knowledge Science, Engineering And Management, Springer, 2006, Pp. 215--228.

J.-P. Bao, J.-Y. Shen, X.-D. Liu, Q.-B. Song, (2003), "*A Survey On Natural Language Text Copy Detection*", Journal Of Software 14 (10) (2003) 1753-1760.

M. Elhadi, A. Al-Tobi, (2008) "*Use of Text Syntactical Structures In Detection Of Document Duplicates"*, In: Digital Information Management, 2008. ICDIM 2008. Third International Conference On, Ieee, 2008, Pp. 520-525.

M. Mišic, (2017) "*Improving source code plagiarism detection systems*," PhD thesis, School of Electrical Engineering, University of Belgrade, Belgrade, 2017. (in Serbian)

M. Potthast, A. Barron-Cede~No, B. Stein, P. Rosso, (2011) "*Cross-Language Plagiarism Detection*", Language Resources And Evaluation 45 (1) (2011) 45--62. 36. A.

M. Rahman, W. P. Yang, T. W. Chow, S. Wu, A Exible (2007) "*Multi-Layer Self-Organizing Map For Generic Processing Of Tree-Structured Data*", Pattern Recognition 40 (5) (2007) 1406--1424.

M. S. Anderson, N. H. Steneck, (2011) "*The Problem of Plagiarism*", In: Urologic Oncology: Seminars and Original Investigations, Vol. 29, Elsevier, 2011, pp. 90-94.

M. J. Mišić, J. Ž. Protić and M. V. Tomašević, (2017) "*Improving source code plagiarism detection: Lessons learned,"* 2017 25th Telecommunication Forum (TELFOR), 2017, pp. 1-8, doi: 10.1109/TELFOR.2017.8249481.

N. Charya, K. Doshi, S. Bawkar, R. Shankarmani, (2015) "*Intrinsic Plagiarism Detection In Digital Data,"* International Journal of Innovative and Emerging Research in Engineering, Volume 2, Issue 3, e-ISSN: 2394 – 3343, p-ISSN: 2394 – 5494

O. Karnalim and L. Sulistiani, (2018) "*Which Source Code Plagiarism Detection Approach is More Humane?*" 2018 9th International Conference on Awareness Science and Technology (iCAST), 2018, pp. 291-296, doi: 10.1109/ICAwST.2018.8517170.

O. Karnalim, Simon and W. Chivers, (2019) "*Similarity Detection Techniques for Academic Source Code Plagiarism and Collusion: A Review*," 2019 IEEE International Conference on Engineering, Technology and Education (TALE), 2019, pp. 1-8, doi: 10.1109/TALE48000.2019.9225953.

PLAG tool demonstration image are taken from this URL: http://www.ics.heacademy.ac.uk/resources/assessment/plagiarism/demo_jplag.html

R. Yerra, Y.-K. Ng, "*A Sentence-Based Copy Detection Approach For Web Documents*", In: International Conference On Fuzzy Systems And Knowledge Discovery, Springer, 2005, Pp. 557--570.

Resnik, Et Al., (1999) "*Semantic Similarity in A Taxonomy: An Information-Based Measure and Its Application to Problems Of Ambiguity In Natural Language*", J. Artif. Intell. Res.(Jair) 11 (1999) 95--130.

S. M. Alzahrani, N. Salim, (2009) "*On The Use Of Fuzzy Information Retrieval For Gauging Similarity Of Arabic Documents*", In: Second International Conference on the Applications Of Digital Information And Web Technologies, 2009. ICADIWT'09, IEEE, 2009, pp. 539-544.

S. M. Zu Eissen, B. Stein, M. Kulig, (2007) "*Plagiarism Detection without Reference Collections*", In: Advances in Data Analysis, Springer, 2007, Pp. 359--366.

Shen Yang, Li Shu-Chen, Tian Chen-Geng, Cheng Ming, (2009) "*Research on Anti-Plagiarism System and the Law of Plagiarism*", 978-0-7695-3557-9/09 $25.00 © 2009 IEEE

Simon, Trina Myers, Dianna Hardy, and Raina Mason (2019) "*Variations on a Theme: Academic Integrity and Program Code*". In Twenty-First Australasian Computing Education Conference (ACE'19), Sydney, NSW, Australia. ACM, New York, NW, USE, 8 pages.

T. W. Chow, M. Rahman, (2014) "*Multilayer Som with Tree-Structured Data for Efficient Document Retrieval and Plagiarism Detection*", IEEE Transactions on Neural Networks 20 (9) (2009) 1385- -1402

Tapan P. Gondaliya, Dr. Hiren D. Joshi, Hardik Joshi (2014), "*Source Code Plagiarism Detection 'Scpdet': A Review"*, International Journal of Computer Applications (Ijca), Volume 105, No. 17, ISSN: 0975-8887 (Online) pp. 18 - 25, November 2014

DOI: http://doi.org/10.24086/cocos2022/paper.658

V. Mitra, C.-J. Wang, S. Banerjee (2007), "*Text Classification: A Least Square Support Vector Machine Approach*", Applied Soft Computing 7 (3) (2007) 908-914.

Xi Xu, Ming Fan, Ang Jia, Yin Wang, Zheng Yan, Qinghua Zheng, Ting Liu (2020). "Revisiting the Challenges and Opportunities in Software Plagiarism Detection". 27th IEEE International Conference on Software Analysis, Evolution and Reengineering Early Research Achievement Track (SANER-ERA), 2020, (CCF B)

Salton, G., Wong, A., & Yang, C.S. (1975). A Vector Space Model for Automatic Indexing. Communications of the ACM 1975, 18(11), 613-620.

Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., & Harshman, R. (1990). 193 Indexing by Latent Semantic Analysis. Journal of the American Society for Information Science, 41(6), 391-407.

Jonathon Shlens (2003). A Tutorial on Principal Component Analysis. Derivation, Discussion and Singular Value Decomposition.

Comon, P. (1994). Independent Component Analysis, A New Concept? Signal Processing, 36(3), 287-314.

Hyvarinen, A., Karhunen, J., & Oja, E. (2001). Independent Component Analysis. John Wiley & Sons, New York.

Kolenda, T., Hansen, L., & Larsen, J. (2001). Signal Detection Using ICA: Application to Chat Room Topic Spotting. In ICA, San Diego, CA, USA, 540-545.

Forsythe, G.E., Malcolm, M.A., & Moler, C.B. (1977). Computer Methods for Mathematical Computations (Chapter 9: Least Squares and the Singular Value Decomposition). Englewood Cliffs, NJ: Prentice Hall.

Harshman, R.A. (1970). Foundations of the PARAFAC Procedure: Models and Conditions for an "Explanatory" Multi-modal Factor Analysis. UCLA Work Papers Phonetics, 16, 86.

Harshman, R.A. & Lundy, M.E. (1984a). The PARAFAC Model for Three-way Factor Analysis and Multi-dimensional Scaling. In H.G. Law, C.W. Snyder, Jr., J.A. Hattie, and R.P. McDonald (Eds.). Research Methods for Multimode Data Analysis, Praeger.

Harshman, R.A. & Lundy, M.E. (1984b). Data Preprocessing and the Extended PARAFAC Model. In H.G. Law, C.W. Snyder, Jr., J.A. Hattie, & R.P. McDonald (Eds.). Research Methods for Multimode Data Analysis, Praeger.

Park, L.A.F., Ramamohanarao, K., & Palaniswami, M. (2005b). A Novel Document Retrieval Method Using the Discrete Wavelet Transform. ACM Transactions on Information Systems, 23(3), 267-298.

Needleman, S.B., & Wunsch, C.D. (1970). A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. Journal of Molecular Biology, 48, 443-453.

Hirschberg, D. S. (1977). Algorithms for the Longest Common Subsequence Problem. Journal of the ACM (JACM), 24(4), 664 - 675.

Cormen, T.H., Leiserson, C.E. & Rivest, R.L. (1990). Introduction to Algorithms, MIT Press.

Gonnet, G. H. & Baeza-Yates, R. (1991). Handbook of Algorithms and Data Structures (Second Edition),Addison-Wesley

Smith, T.F. & Waterman, M.S. (1981). Identification of Common Molecular Subsequences. Journal of Molecular Biology, 147, 195-197.

M. Mozgovoy (2006). Desktop tools for offline plagiarism detection in computer programs. Informatics in Education, 5(1):97–112, 2006.

T. Lancaster and F. Culwin (2005). Classifications of plagiarism detection engines. Italics, 4(2), 2005.

K. Verco and M. Wise (1996a). Plagiarism a la mode: A comparison of automated systems for detecting suspected plagiarism. The Computer Journal, 39(9):741–750, 1996.

Kristina L. Verco and Michael J. Wise. (1996b) Software for detecting suspected plagiarism: comparing structure and attribute-counting systems. In Proceedings of the first Australian conference on Computer Science Education, pages 81–88. ACM Press, 1996.

K. Ottenstein (1976a) An algorithmic approach to the detection and prevention of plagiarism. ACM SIGCSE Bulletin, 8(4):30–41, 1976.

K. Ottenstein (1976b) A program to count operators and operands for ANSI FORTRAN modules. Computer Sciences Report TR 196, Purdue University, 1976.

M. Halstead (1972) Natural laws controlling algorithm structure? ACM SIGPLAN Notices, 7(2):19–26, 1972.

M. Halstead (1977) Elements of Software Science (Operating and programming systems series). Elsevier Science, New York, USA, 1977.

S. Robinson and M. Soffa (1980) An instructional aid for student programs. In SIGCSE '80: Proceedings of the Eleventh SIGCSE Technical Symposium on Computer Science Education, pages 118–129, New York, NY, USA, 1980. ACM.

G. Rambally and M. Sage (1990) An inductive inference approach to plagiarism detection in computer program. In In Proceedings of the National Educational Computing Conference, pages 22–29, Nashville, Tennesse, 1990.

G. Whale (1990) Identification of program similarity in large populations. The Computer Journal, 33(2):140–146, 1990.

S. Schleimer, D. Wilkerson, and A. Aiken (2003). Winnowing: local algorithms for document fingerprinting. In SIGMOD '03: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, pages 76–85, New York, NY, USA, 2003. ACM.

Koschke, R. (2007). Survey of Research on Software Clones. In Proc. of Dagstuhl Seminar 06301: Duplication, Redundancy, and Similarity in Software, 962.

J. Donaldson, A. Lancaster, and P. Sposato (1981) A plagiarism detection system. SIGCSE Bulletin, 13(1):21–25, 1981.

M. Joy and M. Luck (1999). Plagiarism in programming assignments. IEEE Transactions on Education, 42(1):129–133, 1999.

Mozgovoy, M. and Frederiksson, K. and White, D.R. and Joy, M.S. and Sutinen, E. (2005). "Fast Plagiarism Detection System". In: String Processing and Information Retrieval: 12th International Conference (SPIRE 2005). LNCS (3772). Springer, London, pp. 267-270.

L. Prechelt, G. Malpohl, and M. Philippsen (2002). Finding plagiarisms among a set of programs with JPlag. Journal of Universal Computer Science, 8(11):1016–1038, 2002.

G. Cosma and M. Joy (2006). Source-code plagiarism: A U.K academic perspective. Research Report No. 422, Department of Computer Science, University of Warwick, 2006.

M. Mozgovoy, S. Karakovskiy, and V. Klyuev (2007a). Fast and reliable plagiarism detection system. In Frontiers in Education Conference - Global eEngineering: Knowledge without Borders, pages S4H–11–S4H–14, 2007

M. Wise (1996) YAP3: improved detection of similarities in computer program and other texts. SIGCSE Bulletin, 28(1):130–134, 1996.

P. Clough (2000). Plagiarism in natural and programming languages: An overview of current tools and technologies. Technical Report CS-00-05, University of Sheffield, 2000.

A. Ahtiainen, S. Surakka, and M. Rahikainen (2006). Plaggie: Gnu-licensed source code plagiarism detection engine for Java exercises. In Baltic Sea '06: Proceedings of the 6th Baltic Sea Conference on Computing Education Research, pages 141–142, New York, NY, USA, 2006. ACM.

M. Mozgovoy (2007b). Enhancing Computer-Aided Plagiarism Detection. Dissertation, Department of Computer Science, University of Joensuu, Department of Computer Science, University of Joensuu, P.O.Box 111, FIN-80101 Joensuu, Finland, November 2007.

B. Baker (1995a). Parameterized pattern matching by Boyer-Moore type algorithms. In SODA '95: Proceedings of the Sixth Annual ACM-SIAM

DOI: http://doi.org/10.24086/cocos2022/paper.658

Symposium on Discrete Algorithms, pages 541–550, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics.

B. Baker (1995b). On finding duplication and near-duplication in large software systems. In WCRE '95: Proceedings of the Second Working Conference on Reverse Engineering, pages 86–95, Washington, DC, USA, 1995. IEEE Computer Society.

D. Gitchell and N. Tran (1999). Sim: a utility for detecting similarity in computer programs. SIGCSE Bulletin, 31(1):266–270, 1999

A. Amir, M. Farach, and S. Muthukrishnan (1994). Alphabet dependence in parameterized matching. Information Processing Letters, 49(3):111–115, 1994

R. Idury and A. Schaffer (1994). Multiple matching of parameterized patterns. In CPM '94: Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching, pages 226–239, London, UK, 1994. Springer-Verlag.

L. Salmela and J. Tarhio (2006). Sublinear algorithms for parameterized matching. Lecture Notes in Computer Science, 4009/2006:354–364, 2006

K. Fredriksson and M. Mozgovoy (2006). Efficient parameterized string matching. Information Processing Letters, 100(3):91–96, 2006

A. Flint, S. Clegg, and R. Macdonald (2006). Exploring staff perceptions of student plagiarism. Journal of Further and Higher Education, 30:145–156, 2006.

B. Belkhouche, A. Nix, and J. Hassell (2004). Plagiarism detection in software designs. In ACM-SE 42: Proceedings of the 42nd Annual Southeast Regional Conference, pages 207–211, New York, NY, USA, 2004. ACM

S. Dumais, G. Furnas, T. Landauer, S. Deerwester, and R. Harshman (1988). Using latent semantic analysis to improve access to textual information. In CHI '88: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pages 281–285, New York, NY, USA, 1988. ACM.

J. Zhu, Z. Wu, Z. Guan, and Z. Chen (2015), "Appearance similarity evaluation for Android applications", in Seventh International Conference on Advanced Computational Intelligence, IEEE, 2015, pp. 323–328, I S B N: 978-1-4799-7257-9. D O I: 10.1109/ICACI. 2015.7184722.

L. Jiang, G. Misherghi, Z. Su and S. Glondu (2007), "DECKARD: Scalable and Accurate Tree-Based Detection of Code Clones," 29th International Conference on Software Engineering (ICSE'07), 2007, pp. 96-105, doi: 10.1109/ICSE.2007.30.

S. Engels, V. Lakshmanan, and M. Craig (2007). Plagiarism detection using feature-based neural networks. ACM SIGCSE Bulletin, 39(1):34–38, 2007.

M.L. Kammer (2011). Plagiarism detection in Haskell programs using call graph matching. PhD thesis, 2011

J. Hage, B. Vermeer, and G. Verburg (2013). Research paper: Plagiarism Detec tion for Haskell with Holmes. In M. C. J. D. van Eekelen, E. Barendsen, P. B. Sloep, and G. C. van der Veer, editors, Proceedings of the 3rd Com puter Science Education Research Conference, CSERC 2013, Arnhem, The Netherlands, April 04 - 05, 2013, pages 19–30. ACM, 2013.

A. Eppa and A. Murali (2022), "Source Code Plagiarism Detection: A Machine Intelligence Approach," 2022 IEEE Fourth International Conference on Advances in Electronics, Computers and Communications (ICAECC), 2022, pp. 1-7, doi: 10.1109/ICAECC54045.2022.9716671.

DOI: http://doi.org/10.24086/cocos2022/paper.658

TABLE I
COMPARATIVE ANALYSIS

Reference	Weaknesses	Scope of the work
Hayden Cheers, et. al 2020	Plagiarism checking methods based on code (SCPDTs) are not immune to pervasive duplication.	In this approach, optimization and formalization of modifications are carried out. Conducting a more in-depth evaluation in order to make a firm decision. if it is capable of detecting plagiarism that is true- or false-positive, true- or false-negative, and then making it available to the public.
Marko J. Mišic, et.al. 2017	Network with much more over 50 nodes become illegible soon. The usability of the resulting network visualisation is also influenced by the node arrangement.	A system like this should include a variety of similarity detection algorithms. In order to increase effectiveness and decrease execution speed, implementation should indeed be applied. Intelligent reports can be generated using machine learning approaches.
Dieter Pawelczak, 2018	Students use code obfuscation to avoid detection by the system. Many lecturers have been criticized of pushing pupils to be using code found on websites like Stack Overflow. Additionally, reading and comprehending another person's source code, as well as teamwork in programming issues, is an important indicator of a student's competence.	Overall, they should improve programming quality by discussing the students' various approaches and, if necessary, presenting the improvements that their programmes have been able to achieve.
Oscar Karnalim, Lisan Sulistiani, 2018	They are linked via the human-annotated dataset, but only in a roundabout way. As a result of the presence of false results, cognitive dissonance like this could complicate the accusation process.	It is intended to use an aspect-oriented evaluation mechanism to assess the human-oriented effectiveness of the most frequently used features in terms of detecting source code plagiarism. This is also proposed to unify way people suspected republish software, using or not using a statistical model.
Oscar Karnalim, Simon, William Chivers, 2021	A comparison would undoubtedly be a time-consuming task.	To use an individual data collection, evaluate the results of all the other strategies given.
Dirson Santos de Campos, Deller James Ferreira, 2021	It is becoming important to construct the recommended alternative techniques, which are particularly adept reducing compares for identifying duplication based on correlation made inside code (the very same pupil) or between university learners.	The goal is to conduct a sentiment analysis of the students in relation to the difficulties that exist in programming when using advanced text mining techniques developed by others.
Hayden Cheers, et.al. 2021	This research is focused solely on the assessment of SCPDTs. There really are, nevertheless, a slew of alternative techniques that assess system software similarities across disciplines.	PDGs were robust to outliers of significantly changing software, according to the tests. It would be exciting to look at conceptual, and maybe behaviour patterns, approaches of assessing code comparability in the context of ubiquitous alterations in deeper levels.
Xi Xu, 2020	The present scheme is confined to achieving the following extremely desirable prerequisites: 1) Asymmetric handling, 2) Partially Imitation Identification, 3) Software Misinformation Resistant, 4) Errors in reading of Performance Accuracy, and 5) Flexibility to Success Factors and project success Programs	The main research challenges were identified, and numerous potential research directions were proposed to inspire future work.
Farhan Ullah, et.al. 2021	The categorization of several of the themes was wrong, that had a substantial influence on the entire categorization. To solve this problem, the SMOTE method was utilised to left of the mpp the lesser visible results in terms of quality assurance.	Despite the fact that the proposed method is new and has the potential to produce promising results, certain constraints existed that may have an impact on classification performance?
Mayank Agrawal, 2020	A Java-based tool was proposed that will aid in determining the similarity of source codes written in the Java programming language.	Future projects will include work in other languages such as C, Python, R, and Julia. This will be done so that any code written in these languages can be tested using the proposed algorithm.

DOI: http://doi.org/10.24086/cocos2022/paper.658

TABLE I
COMPARATIVE ANALYSIS

Reference	Weaknesses	Scope of the work
Tony Ohmann, 2014	The optimal parameter values for a variety of algorithms can be variable when implemented in PIY, depending on the dataset. After adding a large number of new documents, re-clustering can take days or even weeks.	In the future, user studies will be conducted to investigate the final form of the output produced by PIY.
Farhan Ullah, 2018	The LSA is being used to estimate semantic relatedness across scripting languages, which is then employed in from before the steps, grading, and SVD for extracting buzzwords from C++ and Java source programs.	To improve calculation results, the plagiarism detection tool can be embedded within the LSA. The resemblance exactly with the LSA in the multiple content codes were investigated using the Sherlock plagiarism detection programme.
Hayden Cheers, 2021	BPlag, as a SCPDT, lacks efficiency due to the significantly more complex implementation of this approach. Because it is primarily a graph-based system. A longer run time will be needed.	BPlag is susceptible to value-injecting transformation investigation, which is an appropriate way to mitigate the impact of such transformations. It would be interesting to investigate whether it is possible to combine BPlag with structural and semantic tools in order to obtain a more detailed view of the ways in which the two programmes may be similar.
Farhan Ullah, et.al. 2018	Java, C, C++, C#, and Python were among the dialects included in database. This allowed for the incorporation of certain duplication detection method into the planned research, leading in a far more accurate assessment.	The purpose was to examine source code blocks for linguistic resemblance using the Sherlock plagiarism checking software in an experimental.

DOI: http://doi.org/10.24086/cocos2022/paper.658

DOI: http://doi.org/10.24086/cocos2022/paper.658